Protocol Reverse Engineering: Challenges and Obfuscation

J. Duchêne[1,3], C. Le Guernic[1,2], E. Alata[3], V. Nicomette[3], and M. Kaâniche[3]

[1] DGA Maîtrise de l'Information, Rennes, France.
julien.duchene@intradef.gouv.fr,colas.le-guernic@intradef.gouv.fr
[2] Laboratory High Security, INRIA team TAMIS, Rennes, France
colas.le-guernic@inria.fr
[3] LAAS-CNRS, Univ. de Toulouse, CNRS, INSA, Toulouse, France
jduchene@laas.fr,ealata@laas.fr,nicomett@laas.fr,kaaniche@laas.fr

Abstract. Reverse engineering of communication protocols is aimed at providing methods and tools allowing to infer a model of these protocols. It is very relevant for many application domains, such as interoperability or security audits. Recently, several tools have been developed in order to automate, entirely or partially, the protocol inference process. These tools rely on several techniques, that are usually tuned and adapted according to the final goal of the reverse engineering task. The aim of this paper is 1) to present an overview of the main challenges related to reverse engineering, and 2) to introduce the use of obfuscation techniques to make the reverse engineering process more complex and difficult in particular to malicious users.

1 Introduction

Communication protocols allow several components to exchange messages in a consistent way. They are widely used in networks and telecommunications domains. A protocol may be published in an open standard or it may be proprietary and thus hidden from a user of the component. Reverse engineering is mainly useful in this second case, in the context of non documented and non standardised close protocols. Protocol reverse engineering consists in inferring a model of communications established between several components.

Samba project is a popular example of protocol reverse engineering [16]. It offers an open-source implementation of *SMB/CIFS* protocols for Linux clients, enabling Linux and Windows systems to interoperate. At its beginning in 1992, this project was mainly based on manual reverse, a tricky and time consuming work whose success is tightly linked to the skills of the analysts. Moreover, keeping pace with protocol evolutions was a real challenge.

Protocol reverse engineering techniques are most of the time classified into two categories: network traces analysis tools, and application execution traces analysis tools. They can also be differentiated according to their inference type: passive or active. While active inference stimulates the system in order to discover or validate information, passive inference is only based on captured data.

During the past decade, several protocol reverse engineering tools have been developed [14]. The main application domains of reverse engineering are presented in Section 2. The contribution of this paper is twofold: i) Section 3 discusses the main open challenges raised in the context of protocol reverse engineering, and ii) Section 4 presents a proposal to make reverse engineering more difficult to perform for attackers, thanks to the use of obfuscation techniques. Finally, Section 5 concludes this paper.

2 Protocol reverse application domains

As presented in the introduction, **interoperability** is one of the domains concerned by reverse engineering. **Network protocols simulation** is another domain, with tools such as *Scriptgen* [13], *RolePlayer* [9], *Replayer* [15] or *Rosetta* [6]. Network simulators are useful to quickly prototype some specific tests of a protocol whereas performing such tests on a real implementation may be tedious or practically impossible. Furthermore, a network simulator may replay communications, in various environments, and possibly adapt them. It is relevant *e.g.*, to analyse a network attack or to develop honeypots that can interact with attackers in order to record and analyse their behaviour.

Software security audits (and associated tools: *Netzob* [3], *Polyglot* [7] or *Tupni* [10]) is another relevant application domain which is closely linked to the previous one. However, its main goal differs: a component is solicited under various scenarios to check whether it correctly handles communications in such scenarios. Thus, a model of the protocol may be used in order to develop smart fuzzers useful for testing robustness of a protocol implementation.

Some tools like *Netzob* [2] again or *Dispatcher* [5, 8] enable **Malware protocol analysis**. Indeed, many malwares use protocols to communicate with third parties. The reverse engineering of these protocols is useful to identify some crucial information regarding the localisation of a botnet master, a date, an imminent attack, attack targets, *etc*, and as a consequence, allows to anticipate an attack occurrence and to react accordingly.

Finally, reverse engineering can also be used to support **network protocol conformance testing**. It consists in checking whether a software correctly implements a network protocol whose specification is known. Reverse engineering enables to get a model of the protocol as implemented by an application, then to check whether this model is compliant with its specification or not.

3 Protocol reverse engineering challenges

Protocol reverse engineering raises several challenges. This section presents the different steps of protocols reverse engineering and their associated challenges.

The preliminary phase of protocols reverse engineering should be dedicated to the identification and characterisation of the environment. Based on this knowledge, an analyst can start the observation step, which consists in setting up some probes for collecting network or application traces. The next step consists

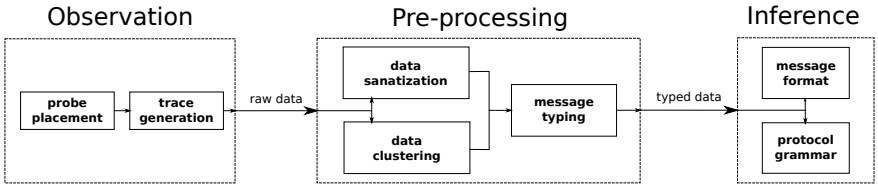

Fig. 1. Protocol reverse engineering steps with associated challenges

in sanitizing these traces in order to obtain relevant messages of the protocol under study. The last step carries out the inference of the message format or of the protocol grammar, from messages obtained at the previous step. These different steps are executed according to an iterative process and the expertise and the intuition of the analyst may largely facilitate some of them. Figure 1 illustrates the steps of the protocol reverse engineering process, and highlights the main challenges associated to each step.

3.1 Observation step

The inference is based on the gathering of a set of traces thanks to the observation of a communication channel. Two challenges are associated to this observation step: **probe placement** and **traces generation**.

Probe placement is essential to capture different messages required to protocol reverse engineering. This may not be obvious if for instance, applications use several protocols to communicate, each protocol on different channels. Furthermore, a protocol under study may be encapsulated into encrypted channels, thus, it may be necessary to implement the probe in the application itself, at the interface of the cryptographic libraries, in order to be able to decrypt the messages for the sake of the analysis. To our knowledge, this challenge is generally not addressed by any tool, only *Netzob* proposes some generic probes [3].

Collected traces quality also depends on observation duration. A too short observation phase may lead to a poor trace, not containing a set of all the possible message sequences. Such a trace may lead to an incomplete inference. Thus, it is important to identify when the trace is sufficiently rich to be exploitable. This depends on components involved in the protocol as well as their communication frequency. In some cases, to obtain more quickly some appropriate traces, the analyst may adopt an active approach, by stimulating the system, e.g. in [3].

3.2 Pre-processing step

The pre-processing step may be all the more difficult to carry out as an analyst could not place the probes ideally. In this case, collected traces may contain irrelevant information that have to be filtered out. Furthermore, messages relevant to a protocol under reverse engineering may be 1) encapsulated into another protocol; 2) split into several packets transferred in multiple exchanges or 3)

observed among messages belonging to other protocols. Thus, a first challenge in this pre-processing step is to correctly sanitize traces in order to reconstruct appropriate messages. This corresponds to **data sanitization**.

When messages are transferred in several packets, another challenge consists in aggregating traces in order to reconstruct messages for the analysis. This challenge concerns both the inference based on network traces, (such as for gathering the TCP segments to obtain a message of an applicative protocol), but also the inference based on application traces, when the analysis of a message is split in several execution traces. Moreover, a trace may contain data related to several messages, thus it is necessary to identify and split corresponding data of each message. These two elements are referred to as **data aggregation**. Data sanitization and aggregation are addressed in [13], but generally left aside.

After their reconstruction, messages are grouped into classes of messages. This clustering phase is required in order to compare the messages that are semantically equivalent. This phase consists in finding a **typing** function allowing, from a message represented as a sequence of bytes, to identify its type. This challenge is addressed by every network based analysis tools except *ReverX* [1], but application based analysis tools generally left it aside.

3.3 Inference step

Message format inference is aimed, from messages of a same type, at identifying their structure whereas **protocol grammar inference** is aimed at recovering, from sequences of typed messages, rules describing their exchanges. It is also important to identify dependencies between different fields of a message or between messages themselves.

The goal of reverse engineering process is to obtain a specification for message format or protocol grammar. This specification is represented by a model. It is necessary to choose a sufficiently expressive model so that it can faithfully reflect the original specification. For instance, some complex message formats or protocol grammars may have a tree or recursive structure. Such a structure means that a message or value of a field depends on other messages or other fields. This dependency is difficult to express with finite state automata for instance.

4 Obfuscation techniques

Our current research work focuses on obfuscation techniques aiming at making protocol reverse engineering more difficult to perform in the context of interoperability or network simulation. Such techniques must be easily plugged to existing solutions. Previous work [12, 4] has focused on designing protocols similar to widely used protocols to bypass network traffic classification. Thus, our objective is to make it difficult, not to say impossible, for an attacker to entirely reverse the protocol in a reasonable time. The goal is, given a protocol specification, to generate multiple instances of obfuscated protocol, for instance, each

one dedicated to a client. This work is orthogonal to cryptographic solutions, generally speaking, it is less robust but faster in its treatment.

To reach our objective, we focus on some challenges presented in the previous section: message typing and the inference itself. Our methodology relies on the following observations: 1) message typing is mostly realized via similarity and 2) inferred models are equivalent to regular languages, e.g., message format only depends on its type. Accordingly, the obfuscated protocols should be more complex than regular protocols. In this case, generalization performed by tools will lead to lots of wrong messages. Moreover, we also propose to ensure that message format depends on the communication context, and to force the dissimilarity between messages originally belonging to the same type.

We plan to adopt algebraic or contextual languages. Most of the time, message format and protocol grammar can both be expressed with formal grammar. Thus, an obfuscation strategy for one of them can be adapted to the second one. For example, the message format "$M \rightarrow$ **<K,V>;**M|**<K,V>**", corresponding to a list of (key, value) (for instance **<K_1,V_1>;<K_2,V_2>;<K_3,V_3>**) is a regular language that can easily be transformed into "$M \rightarrow$ **<K**M**V>|,**", (i.e., **<K_1<K_2<K_3,V_3>V_2>V_1>**) which corresponds to an algebraic language ($a^n b^n$). The inference of this class of languages is more tricky [11]. Moreover, if modifications injected regularly change, according to the previously exchanged message, this means that we enrich the communication with a context which is difficult to reverse. A same message sent by a component at two different moments will then be obfuscated in two different ways.

We are currently prototyping our solution. It corresponds to a stub added to legitimate communicating components. Each message is processed or generated by this stub, which is itself automatically generated by a framework. Actually, generated code is derived from the original specification, applying some transformations on the message format in order to directly handle obfuscated messages. In other words, the stub is able to directly build and parse obfuscated messages. For developers of the components, using the original stub or the obsfuscated one is transparent as the interfaces of message handling functions are identical.

5 Conclusion

This paper presented the main challenges associated to protocols reverse engineering and our first contribution regarding the use of obfuscation techniques to make reverse a quite complex task. We are currently implementing a prototype of an obfuscation framework, based on a contextual grammar that we have designed for this purpose.

References

1. Antunes, J., Neves, N., Verissimo, P.: Reverse Engineering of Protocols from Network Traces. In: 2011 18th Working Conference on Reverse Engineering (WCRE). pp. 169–178. IEEE, New York, NY, USA (2011)

2. Bossert, G., Hiet, G., Henin, T.: Modelling to Simulate Botnet Command and Control Protocols for the Evaluation of Network Intrusion Detection Systems. In: 2011 Conference on Network and Information Systems Security (SAR-SSI). pp. 1–8. IEEE, La Rochelle, France (2011)
3. Bossert, G.: Exploiting Semantic for the Automatic Reverse Engineering of Communication Protocols. phdthesis, Suplec (dec 2014)
4. Bridger, H., Rishab, N., Phillipa, G., Rob, J.: Games Without Frontiers: Investigating Video Games as a Covert Channel. In: Proceedings of the 2016 IEEE European Symposium on Security and Privacy. IEEE European Symposium on Security and Privacy, IEEE (2015)
5. Caballero, J., Poosankam, P., Kreibich, C., Song, D.: Dispatcher: enabling active botnet infiltration using automatic protocol reverse-engineering. In: Proceedings of the 16th ACM conference on Computer and communications security. pp. 621–634. CCS '09, ACM, New York, NY, USA (2009)
6. Caballero, J., Song, D.: Rosetta: Extracting protocol semantics using binary analysis with applications to protocol replay and NAT rewriting. Technical Report CMU-CyLab-07-014, Carnegie Mellon University, Pittsburgh, USA (2007)
7. Caballero, J., Yin, H., Liang, Z., Song, D.: Polyglot: automatic extraction of protocol message format using dynamic binary analysis. In: Proceedings of the 14th ACM conference on Computer and communications security. pp. 317–329. CCS '07, ACM, New York, NY, USA (2007)
8. Caballero Bayerri, J.: Grammar and model extraction for security applications using dynamic program binary analysis. Ph.D. thesis, Carnegie Mellon University, Pittsburgh, PA, USA (2010)
9. Cui, W., Paxson, V., Weaver, N., Katz, R.H.: Protocol-independent adaptive replay of application dialog. In: Proceedings of the 13th Annual Network and Distributed System Security Symposium (NDSS). Internet Society, San Diego, USA (February 2006), `http://research.microsoft.com/apps/pubs/default.aspx?id=153197`
10. Cui, W., Peinado, M., Chen, K., Wang, H.J., Irun-Briz, L.: Tupni: automatic reverse engineering of input formats. In: Proceedings of the 15th ACM conference on Computer and communications security. pp. 391–402. CCS '08, ACM, New York, NY, USA (2008)
11. de la Higuera, C.: Grammatical Inference: Learning Automata and Grammars. Cambridge University Press, New York, NY, USA (2010)
12. Hjelmvik, E., John, W.: Breaking and Improving Protocol Obfuscation. Technical Report 2010-05, Chalmers University of Technology, Gothenburg, Sweden (2010), `http://publications.lib.chalmers.se/cpl/record/index.xsql?pubid=123751`
13. Leita, C., Mermoud, K., Dacier, M.: ScriptGen: an automated script generation tool for Honeyd. In: Computer Security Applications Conference, 21st Annual. pp. 12 pp.–214. IEEE, Tucson, USA (2005)
14. Li, X., Chen, L.: A Survey on Methods of Automatic Protocol Reverse Engineering. In: 2011 Seventh International Conference on Computational Intelligence and Security (CIS). pp. 685–689. IEEE, Hainan, China (2011)
15. Newsome, J., Brumley, D., Franklin, J., Song, D.: Replayer: automatic protocol replay by binary analysis. In: Proceedings of the 13th ACM conference on Computer and communications security. pp. 311–321. CCS '06, ACM, New York, NY, USA (2006)
16. Samba Team: Opening windows to a wider world. http://www.samba.org